

DataONE RunManager: a scientific workflow and provenance data management software

ABSTRACT

In this demo, we demonstrate a scientific workflow and provenance data management software – DataONE RunManager that can capture, store, query, visualize, and publish of a single Matlab script or multiple Matlab script runs. The important features of DataONE RunManager software include: (1) there are three forms of data provenance supported by RunManager: prospective provenance, retrospective provenance, and hybrid provenance; (2) for a workflow project, we have multiple provenance graphs consisting of a graph of prospective provenance, a graph of hybrid provenance, a graph of retrospective multiple_scripts_multiple_runs provenance.

KEYWORDS

prospective provenance, hybrid provenance, retrospective provenance, provenance query

ACM Reference format:

. 2016. DataONE RunManager: a scientific workflow and provenance data management software. In *Proceedings of ACM Conference, Chicago, USA, June 2017 (SSDBM'17)*, 6 pages. DOI:

1 INTRODUCTION

Provenance from scripts and runs of scripts plays an important role in software debugging, testing, reliability and sharing. Such provenance traces consist of events that the user is interested in. A considerable amount of research has been done on investigating methods of harvesting provenance information from scripts and runs of scripts, ranging from conventional approaches, e.g. research compendium (folder layouts) and logging to recent provenance tools, such as YesWorkflow

The DataONE RunManager is a scientific workflow and provenance data management software. It can capture, store, query, visualize, and publish of one or multiple Matlab or R script runs. There are three types of provenance supported by RunManager: prospective provenance, retrospective provenance, and hybrid provenance. The prospective provenance is expressed using YW tags. The retrospective provenance captured during a script execution includes information about the script that was run, files that were read or written, and details about the execution environment at the time of execution.

DataONE RunManagers for R and MATLAB to capture runtime file-level provenance information that are interested by the earth science community. A DataONE package includes scripts, a list of input files, and a list of generated files, science metadata that are associated with the run. In addition, a DataONE data package can be efficiently archived so that the past versions of files can

be retrieved for a run in order to investigate previous versions of processing or analysis, support reproducibility, and provide an easy way to publish data products and all files that contributed to those products to a data repository such as the DataONE network.

Then, we show how to produce hybrid provenance by joining prospective and retrospective provenance with the YW URI mechanism. Last but not least, we propose multi_run provenance. From the multi-run provenance, it enables a longitudinal view of a typical real-life scientific workflow that consists of multiple phases. Since computational and data science experiments can often last days, weeks, or even months and often require the execution of multiple scripts or workflows with varying input datasets and parameters, some of these script runs appear as chained together implicitly via intermediate data. Multiple_scripts_multi_runs provenance graphs can be joined on file path, file content, etc.

We use query-based approach for provenance analysis. A query is implemented in Prolog and SQL now. A graph is used to represent the query results. For a workflow project, we have multiple provenance graphs consisting of a graph of prospective provenance, a graph of hybrid provenance, a graph of retrospective multi-run provenance.

The rest of this paper is organized as follows. The three forms of data provenance used in RunManagers are presented in Section 2. Section 3 discusses two example use cases whereas Section 4 concludes the paper.

Related Work. Table 1 presents a representative set of related provenance tools for provenance collection. YesWorkflow provides prospective graph revealing the data dependencies at the conceptual workflow level. URI-template specifies a file path pattern while user-defined log-files capture runtime provenance observables at any level of granularity. The noWorkflow system (NW) [ref] captures code-level retrospective provenance by employing a Python profiling library to obtain Python function call chains and variable assignments. The DataONE RunManagers [ref] overload and thus intercept file I/O operations to automatically capture runtime observables at the file level.

2 PROSPECTIVE PROVENANCE, RETROSPECTIVE PROVENANCE AND HYBRID PROVENANCE

Data provenance is metadata that describes the origin and processing history of a data artifact. In the computational and data sciences, we can distinguish three forms of data provenance, i.e., prospective provenance, retrospective provenance, and hybrid provenance. The prospective provenance describes the general workflow by which data is produced, while the retrospective provenance consists of runtime observables, e.g., names, locations, and contents of files read and written during a script run (coarse-grained observables). Fine-grained provenance can be captured as well, e.g., changes to individual data records, program variables, parameter

Table 1: Various Provenance Tools and Methods Breakdown

	Language: Application / Implementation	Scope	Methods
YesWorkflow	Any / Java	Prospective Provenance	Annotations added by users; language independent
URI-templates	Any / Any	Retrospective Provenance	Templates matching with runtime file locations; language independent
User-defined log-files	Any / Any	User-defined Retrospective Provenance	Exported runtime variables and files provenance by users
noWorkflow	Python / Python	Retrospective Provenance	Runtime function calls, variables, file inputs/outputs, execution environmental information
RunManagers	R / R MATLAB / MATLAB	Retrospective Provenance	Runtime file inputs/outputs, execution environmental information

settings, etc. Hybrid provenance can be defined as a graph containing an alternation of process steps, situated in the context of the entire process structure, and of the specific instances of data elements that have been generated and consumed by executions of those steps. The data elements can be instantiated to document the specific files used in a concrete workflow run.

Listing 1: YW annotation block in C3C4 Matlab script

```
%% Load input: long-term monthly mean precipitation data
% @BEGIN fetch_monthly_mean_precipitation_data
% @in mean_precip @URI
% file:inputs/narr_apcp_rescaled_monthly/
% apcp_monthly_{start_year}_{end_year}_mean.{month}.nc
% @out Rain @AS Rain_Matrix
Rain=zeros(ncols,nrows,12);
for m=1:12
    rncid=netcdf.open(strcat(
        'inputs/narr_apcp_rescaled_monthly/monthly_2000_2010_mean.',
        num2str(m), '.nc'), 'NC_NOWRITE');
    rvid=netcdf.inqVarID(rncid, 'apcp_monthly_mean');
    Rain(:, :, m)=netcdf.getVar(rncid, rvid);
    netcdf.close(rncid)
end
% @END fetch_monthly_mean_precipitation_data
```

YW and RunManagers that are used by DataONE provenance team are used for demo. YesWorkflow (YW) is a prospective provenance tool allowing users to easily recover high-level workflow models latent in scripts using simple user-annotations embedded as script comments [ref]. List 1 shows a code snippet from a MATLAB script with YW annotations: Here, the @begin and @end tags are used to mark code blocks (i.e., processing steps in the YW model); @in and @out tags are used to define the dataflow between blocks; and the @uri template (which is paired with an @in tag) is used to define a where the data file is located. Note that @uri templates define a metadata pattern via so-called template variables (here: *start_year*, *end_year*, and *month*). In this way, the script author can document the folder structure and file-naming conventions used by the script. List 2 shows a YW annotation block in a R script.

Listing 2: YW annotation block in one OHIBC R script

```
# @BEGIN output_estimate_status_file_by_region_since_1980
# @param year_span
```

```
# @param dir_goal
# @in area_protected_total_file @URI
file:{dir_goal}/output/area_protected_total.csv
# @in status_file_handle
# @out status_df_data
status_df <- read.csv(file.path(dir_goal, 'output',
    'area_protected_total.csv')) %>%
    select(rgn_id, year, lsp_status) %>%
    filter(year > max(year) - year_span)
# @END output_estimate_status_file_by_region_since_1980

# @BEGIN write_status_to_csv_file
# @in status_df_data
# @out lsp_status_file @URI
file:{dir_goal}/output/lsp_status.csv
write.csv(status_df, status_file)
# @END write_status_to_csv_file
```

RunManagers can transparently capture retrospective provenance from MATLAB and R script executions. DataONE RunManagers overload and thus intercept file I/O operations to automatically capture runtime observables at the file level. DataONE provides two implementations: the recordr R package and the matlab-dataone toolbox. The provenance captured during a script execution includes information about the script that was run, the files that were read or written, and details about the execution environment at the time of execution.

DataONE RunManagers support the following APIs: record, startRecord, endRecord, listRuns, deleteRuns, viewRun(s), traceRuns for R, plotRuns for R and publishRun in order to capture, search, archive, visualize, and reproduce a script run. The captured retrospective provenance are populated to a relational provenance database which has these tables for execution metadata, file metadata, tag, module dependencies for MATLAB and provenance relationships for R. Meanwhile, the captured retrospective provenance can be exported as provenance facts that can be used by logical rules. Retrospective provenance graphs can be rendered from the exported provenance facts. Then, a multiple_scripts_multiple_runs provenance graph can be produced by executing a Prolog query¹ which consists of four Prolog queries

¹https://github.com/yesworkflow-org/yw-idcc-17/blob/master/OHIBC_Howe.Sound.project/multi_runs_retrospective_provenance_queries/queries/render_rm_graph_upstream_of_file.q1.sha256.sh

using the exported provenance facts. A example set of logical rules are highlighted in List 3. The joining operation is based on file content (hash code). For the recursive rules `rm.file.downstream` computes all upstream files for the given file `Sha256_2`. It has two base cases: one is the file object is itself downstream file object and the other is immediate derived relationship.

Listing 3: Prolog rules for multiple scripts multiple runs provenance

```
:- table rm_file_immediately_downstream/2.
rm_file_immediately_downstream(Sha256_1, Sha256_2) :-
    exec_used_files(_, E, _, _, Sha256_1, _, _, _),
    exec_generated_files(_, E, _, _, Sha256_2, _, _, _).

# Base case 1.
:- table rm_file_downstream/2.
rm_file_downstream(Sha256_1, Sha256_2) :-
    Sha256_1 = Sha256_2.

# Base case 2.
rm_file_downstream(Sha256_1, Sha256_2) :-
    rm_file_immediately_downstream(Sha256_1, Sha256_2).

# Recursive part.
rm_file_downstream(Sha256_1, Sha256_2) :-
    rm_file_immediately_downstream(Sha256_1, S),
    rm_file_downstream(S, Sha256_2).

# The rule gv_nodes__files__upstream_of_file_sha256
  computes all file objects which has a given hashcode
  (DataSha256).
gv_nodes__files__upstream_of_file_sha256(_, DataSha256) :-
    rm_file_downstream(Sha256, DataSha256),
    filemeta(_, _, FilePath, Sha256, _, _, _, _),
    gv_labeled_node_sha256(Sha256, FilePath),
    fail
;
true.

# The rule
  gv_edges__execution_to_file__upstream_of_file_sha256
  draws edges from an execution object to file objects
  by first finding all upstream file objects for a given
  hashcode (DownstreamDataSha256).
gv_edges__execution_to_file__upstream_of_file_sha256(_,
    DownstreamDataSha256) :-
    rm_file_downstream(DataSha256, DownstreamDataSha256),
    exec_generated_files(_, ExecutionId, _, _, DataSha256, _, _, _),
    gv_unlabeled_edge_sha256(ExecutionId, DataSha256),
    fail
;
true.
```

Hybrid provenance can be derived using logic rules. The approach of creating hybrid provenance is as follows: after a script run is completed, the retrospective provenance information are collected and populated to Sqlite provenance database. Then, Matlab RunManager has a function to export content of tables `filemeta` as a list of input and output file paths that are fed to `yw-matlab` bridge. The bridge will match the URI templates with the exported file list and the logical rule for producing hybrid provenance is shown in List 4.

Listing 4: Prolog rules for computing hybrid provenance

```
:- table data_uri_resource/2.
data_uri_resource(DataId, ResourceURI) :-
    data_resource(DataId, ResourceId),
    resource(ResourceId, ResourceURI).
```

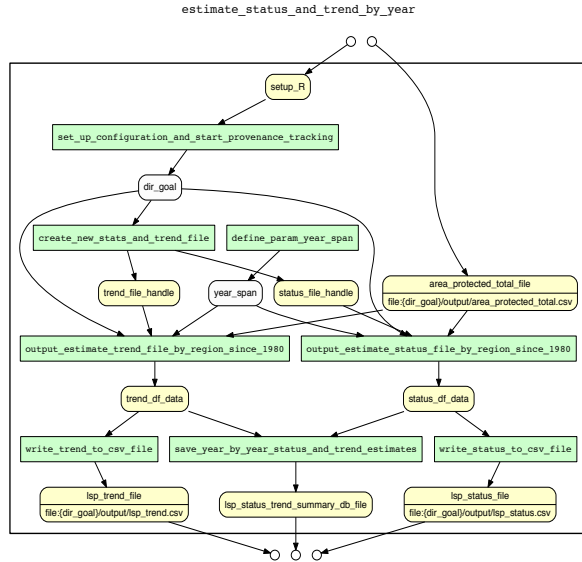
To demonstrate the provenance data dependency analysis using three forms of captured/derived provenance, we have developed both SQL queries and Prolog rule-based queries against various provenance facts. For example: What outputs does the workflow (script) have? Advanced provenance queries can take the form of regular path queries (RPQs), tree-structure queries, or graph queries and show the data lineage hidden in the provenance information. For example: (1) What are the ancestors or descendants of a given output in a prospective provenance graph? (2) What are the ancestors or descendants of a given output in a retrospective- or hybrid- provenance graph? (3) Which possible execution (i.e., dataflow) paths satisfy a given regular path expression? The queries and results are demonstrated in Section 3.

3 EXAMPLE USE CASES

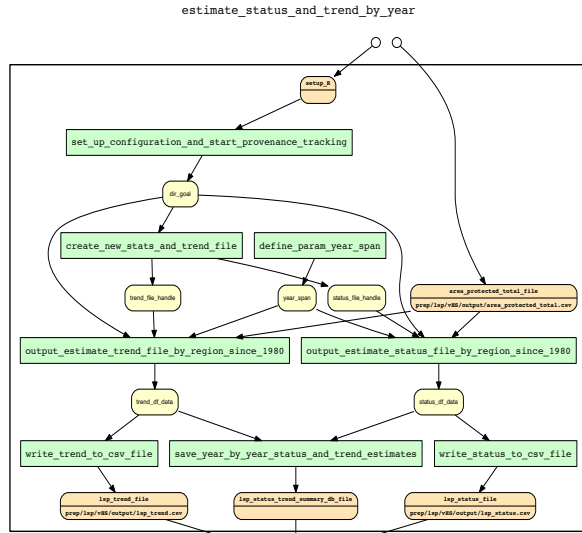
In this demo, two real-life scientific examples are used: (A) a workflow for Ocean Health Index (OHI) for Howe Sound, British Columbia [ref]; (B) a workflow for Carbon3/Carbon4 (C3/C4) soil mapping for North America [ref]. The OHIBC example calculates the region's status based upon percent of protected area within 1 km inland buffer and percent of protected area within 3 nautical mile offshore buffer, compared to a reference point of 30% protected area. The data sources used in the OHIBC workflow include: (1) BC-specific WDPA dataset that was created from WDPA global dataset, then rasterize to BC Albers at 500m resolution; (2) BC Parks, Ecological Reserves, and Protected Areas (PEP) data. The OHIBC workflow creates two outputs: one file is for estimate of status by region since 1980 (`lsp_status.csv`) and the other file is for estimate of trend by region since 1980 (`lsp_trend.csv`). The OHIBC Howe Sound workflow runs in a batch mode.

RunManager can show the prospective provenance and retrospective provenance as graphs for one script. There are eleven R scripts in the OHIBC example. A user can add YW tags to these R scripts to show prospective provenance graphs. Then, the user can start to run the OHIBC example using RunManager. After the run was done, the retrospective provenances (e.g., the execution of each script, used files, generated files including file path, file content, size, and archived file path) are captured and stored in the database. After exporting retrospective provenance as a list of input and output file paths, hybrid provenance can be derived by joining retrospective provenance and prospective provenance facts using shared resource ids. Figure 1 depicts the prospective and hybrid graphs.

RunManager can review the overview graph for all script runs because RunManager have recorded the execution history for the OHIBC example. Hence, using R RunManager provenance database, the `plotRun()` API gives us an overview for the OHIBC Howe Sound workflow example shown in Figure 2. Alternatively, using exported Prolog facts from the provenance database, we can use prolog queries to render a complete `multi_scripts_multi_runs` graph. With



(a) prospective provenance graph



(b) hybrid provenance graph

Figure 1: Complete YW prospective and hybrid provenance graphs for the last step script `estimate_status_and_trend_by_year.R`. The green box represents a computation step; a round yellow square box represents a data element declared in the script via YW tags; an orange box represents a data element which URI is expanded with the runtime file path.

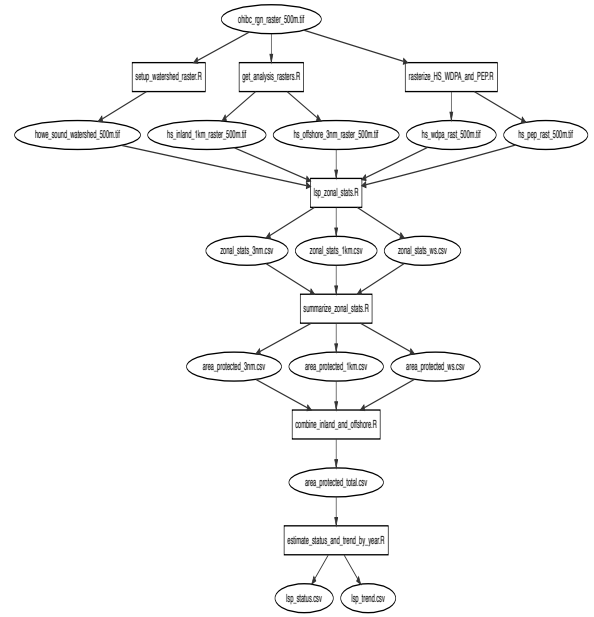


Figure 2: Complete retrospective provenance graph for the OHI Howe Sound workflow example plotted by the API `plotRun()` in DataONE R RunManager

exported Prolog facts, we can use query-based approach to do provenance dependency analysis which will be explained in the next section.

Provenance Query Examples. The first category provenance query is prospective provenance queries can tell us if an output depends a particular input or if an output depends on all of inputs, see the script `estimate_status_and_trend_by_year.R` in Figure 1).

The second category provenance query is hybrid provenance query. In a hybrid provenance graph, the “workflow backbone” is given by the user-declared YW model (prospective provenance) and the execution details are filled in from one or more sources of runtime observables (retrospective provenance). Figure 1 shows a hybrid provenance graph for the last script in the OHIBC example `estimate_status_and_trend_by_year.R`. We can see that the yellow round box changes to orange round box and the file path pattern embedded is expanded with runtime relative file paths. Please visit the web link for prospective and hybrid provenance queries².

The third category provenance query is `multi_scripts_multi_runs` provenance query. RunManager retrospective provenance queries refer to the runtime retrospective provenance obtained from executing multiple scripts, using RunManager. Since the scripts in a project are usually logically connected through sharing some of their inputs and outputs, the corresponding provenance traces are also connected through shared data products. Figure 2 shows the

²https://github.com/yesworkflow-org/yw-idcc-17/tree/master/OHIBC.Howe_Sound.project/hybrid.provenance.queries/queries

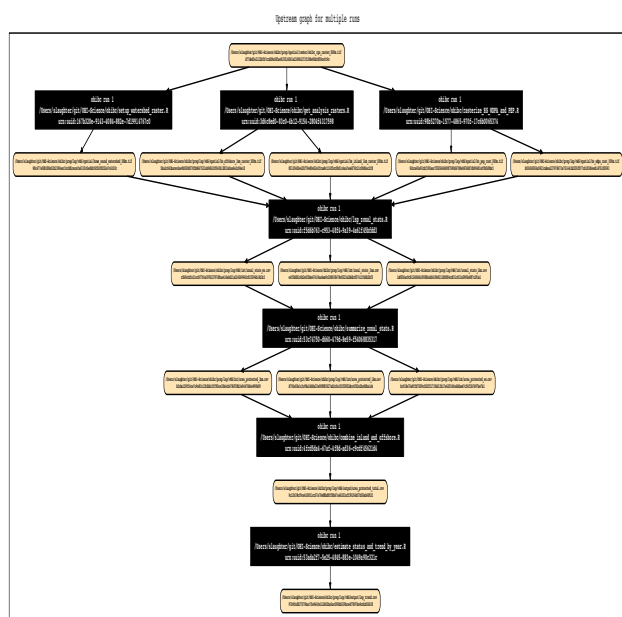


Figure 3: Upstream subgraph that is relevant to “lsp.trend.csv” output for multi_scripts_multi_runs retrospective provenance for the OHIBC Howe Sound example

complete graph for the OHIBC workflow example using the R provenance relational database and Figure 3 shows a upstream retrospective provenance subgraph that is relevant to “lsp.trend.csv” output using exported facts. The multi_scripts_multi_runs query can be found at ³.

The second example script (in MATLAB) produces Carbon 3/Carbon 4 (C3/C4) soil maps for North America using average rain and air temperature monthly data from year 2000 to 2010. There are two versions for C3C4 example. In the version 1, there is one C3C4 example. The captured provenance and query results for C3C4 version is available at ⁴. In order to simulate real-life workflow, we create a version 2 C3/C4 example by splitting one script to three phases: preparing step, examining pixels for grass and generating final results. After the preparing step, a user can run the second script examine pixels for grass for multiple times with different parameters in order to find a proper solution, then writes to the output files. We provide one multi_scripts_multi_runs provenance graphs based on file content in Figure 4 which records a holistic run of the three scripts and the second script was run with two different parameter values (i.e., 2.5 and 5) twice. By comparing these two figures, we can understand that the second script which was run twice produces two versions of files but write to the same file locations.

³https://github.com/yesworkflow-org/yw-idcc-17/tree/master/OHIBC.Howe.Sound.project/multi_runs_retrospective_provenance_queries/queries

⁴<https://github.com/yesworkflow-org/yw-idcc-17/tree/master/examples/C3C4/results>

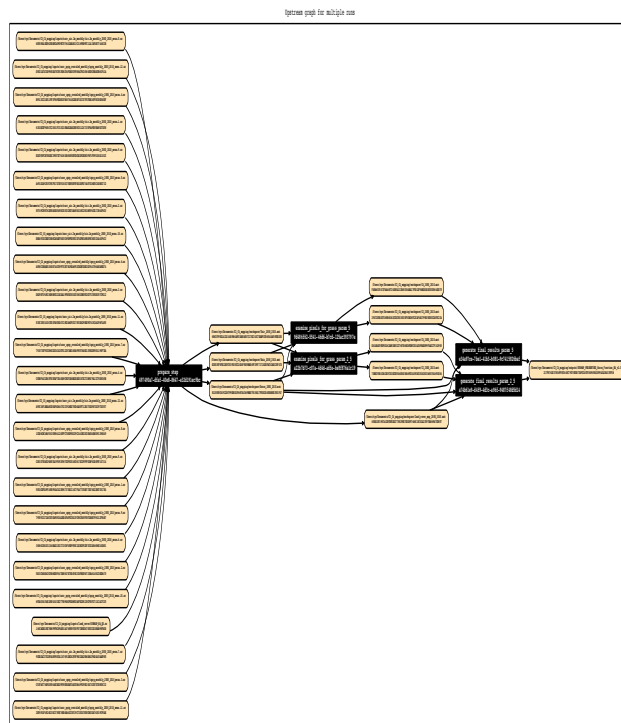


Figure 4: Upstream retrospective provenance subgraph (longitudinal view) that is relevant to one output “SYNMAP_PRESENTVEG_Grass_Fraction_NA_v2.0.nc” for C3C4 version 2 example. In the diagram, an orange yellow box represents a data file and a black box represents an execution (a run).

Reproducibility. DataONE allows to make experiments reproducible by providing two capabilities. First, as we have demonstrated above, DataONE RunManagers are able to capture both prospective and retrospective provenance. Second, DataONE data repository allows users to package workflow results (implemented in any programming languages such as Python, R, MATLAB, SAS, etc.) in a structured DataONE data package and publish to a member node inside a DataONE federation network so that the workflow results (including 2D graphs, 3D graphs, scripts, data, provenance relationship, science metadata describing the experimental methods, authors, geographical coverage, time coverage provided by data providers, etc.) can be displayed on DataONE data repository and be shared or linked with other scientific experiments in a larger scale. It is different from providing reproducibility based on IPython notebook [ref] or a virtual machine, such as a Docker or a VM. DataONE package contains provenance information (prov:used, prov:wasGeneratedBy, or prov:wasDerivedFrom) to describe the relationships between digital objects and datapackages for reproducibility, instead of only encapsulating digital objects.

4 CONCLUSIONS

Provenance is a useful component of the software development cycle, either to benefit researcher themselves or documenting and sharing code for use by future investigators. For the researcher, the quality of data produced by script runs can be assessed by identifying source data and verifying their quality.

Many provenance tools have been developed to capture, store, query and visualize the provenance produced by scripts and runs of scripts. But there is not enough investigation on what kind of provenance queries can be answered from the provenance information at prospective and retrospective provenance levels. Our research on prospective, hybrid and retrospective provenance queries allow researchers to use simple queries and a subset of advanced queries to query across YW model and various retrospective provenance sources to produce interesting provenance artifacts.

With our DataONE RunManager demo, we showcase how RunManager can capture, store, query, visualize, and publish prospective, retrospective and hybrid provenance. Moreover, we demonstrated that provenance queries across heterogeneous provenance sources can yield a more complete and comprehensible picture of data provenance. We also think provenance describing datasets can facilitate data discovery for others.

REFERENCES